Infusing Software Engineering Technology into Practice at NASA

Thomas Pressburger
NASA Ames Research Center
Moffett Field,CA
Tom.Pressburger@nasa.gov

Michael Hinchey
NASA Goddard Space Flight Center
Greenbelt, MD
Michael.G.Hinchey@nasa.gov

Martin S. Feather
Jet Propulsion Laboratory
California Institute of Technology,
Pasadena, CA
Martin.S.Feather@jpl.nasa.gov

Lawrence Markosian
QSS Group, Inc.
NASA Ames Research Center
Moffett Field, CA
lzmarkosian@email.arc.nasa.gov

Abstract

We present an ongoing effort of the NASA Software Engineering Initiative to encourage the use of advanced software engineering technology on NASA projects. Technology infusion is in general a difficult process yet this effort seems to have found a modest approach that is successful for some types of technologies. We outline the process and describe the experience of the technology infusions that occurred over a two year period. We also present some lessons from the experiences.

1. Introduction

Many obstacles impede the infusion of software engineering research results into the development community. Practitioners cannot readily identify the emerging techniques that may benefit them, and cannot afford to risk time and effort evaluating and trying out new techniques while there is uncertainty as to whether they will work for them [1].

This paper describes an ongoing effort conducted by a software engineering research infusion team established by NASA's Software Engineering Initiative to help overcome obstacles to research infusion. The team first identifies and assesses software engineering research relevant to NASA's software development activities. This includes research products from NASA research, NASA-funded research at universities and in industry, and research products external to NASA, which includes leading-edge commercial products.

Next, the team identifies channels to reach the NASA software practitioners who might benefit from these products. These channels are used to publicize the research techniques among NASA and its contractors' software development teams. Then, collaborations are brokered between NASA software engineers and the technology providers. That is, guidance is provided during the proposal development process. Proposals, prepared by the intended users, rather than the technology providers, are evaluated by the research infusion team.

Winning proposals are funded in part by dedicated NASA funding. This funding is for technology introduction risk reduction: licenses, training, and reporting. During the period of the collaboration (typically six months), the research infusion team tracks progress and intervenes, when needed, to help overcome obstacles. Finally, the team extracts lessons learned to sharpen the following year's research infusion effort [2].

2. Infusions to date

To date, 14 research infusion collaborations have been funded, exploiting a wide range of software engineering technologies oriented towards software assurance (this focus, since broadened, was initially part of the team's charter, and a requirement of our funding organization).

Technology categories have included: requirements specification and analysis, software architecture and evaluation, source code analysis and defect detection, and software process improvement. Target projects have included flight software, ground software, Space Station payload software and Space Shuttle software.

Sections 2.1 to 2.9, inclusive, briefly describe each of the nine collaborations to date that have completed and submitted their final reports, along with that collaboration's objectives, what happened, impact on the project, and success criteria. For more details on each of them, see http://ti.arc.nasa.gov/researchinfusion/ Section 3 discusses criteria on which we evaluate our success, and Sections 4 and 5 discuss lessons learned and conclusions.

2.1 ARC: "On Orbit Software Analysis" using C Global Surveyor

This project applied the source code analysis tool C Global Surveyor (CGS), a research tool developed within the Automated Software Engineering group at NASA Ames (ARC) under the Intelligent Systems program of Computing, Information, and Computing Technologies, to a payload software module for the International Space Station (ISS). The tool analyzes C programs to find dead code and memory access errors: de-references of null pointers and out-of-bounds array accesses, and in some cases uninitialized variables. The main benefits expected of applying the tool were finding errors in the software and to give feedback to improve the tool.

The tool reports on the code by classifying operations as green, orange, or red. Green operations never result in a runtime error of the above types. Red operations always result in a runtime error. Orange operations are those for which the tool cannot determine one way or the other whether that operation would cause a runtime error (commonly referred to as "warnings"). An issue with such analysis tools is their scalability and the precision of their analysis. CGS was designed to run quickly on relatively large amounts of code and to be precise about green operations; that is, it categorizes relatively few error-free operations as orange. It is probably less precise, though much faster, than PolySpace Verifier, another static analysis tool, about red operations; that is, operations that always cause errors might be classified by CGS as orange. The designers of CGS claim that its purpose is to do a complete coverage analysis of a software system to quickly narrow down the operations that need to be analyzed or tested further to determine whether they can cause an error. This follows because it was designed to be precise about which operations are green; thus, the amount of code for which further study is required will be minimized.

The research infusion team had somewhat mischaracterized CGS's purpose as to flag errors in software, which requires the tool to be more precise about which operations are red. CGS had been applied to, and specialized in some ways for, Mars Pathfinder software and achieved 80% precision on it; that is, 80% of the operations were classified as red or green. This collaboration was something of an experiment to see if the tool could provide benefit for the analysis of other flight software.

The tool turned out to be about 50% precise on the module, as it had a much different architecture from that of Mars Pathfinder. If the tool were enhanced to deal with certain features of the C language and the application, the precision would have been about 90%. The project found its user interface cumbersome.

There were three important positive outcomes from the collaboration. First, dead code and an uninitialized variable were found in the module. Second, feedback was given to the CGS developers about new capabilities that the tool required to analyze certain features of C and handle this flight software. Third, serendipitously, because of his involvement in the collaboration, one of the CGS developers decided to apply another tool to the module which pinpointed a memory leak that had been suspected by the project.

2.2 GSFC: "GSFC FSB Application of Perspective-Based Inspections"

The goal of this collaboration was to produce Flight Software Branch (FSB) process standards for software inspections which could be used across three new missions within the FSB. The standard was developed by Dr. Forrest Shull (Fraunhofer Center for Experimental Software Engineering, Maryland) using the Perspective-Based Inspection approach, (PBI research has been funded by NASA Software Assurance Research Program (SARP) http://www.ivv.nasa.gov/forresearchers/osmasarp/osmasarp.php), then tested on a pilot Branch project. Because the short time scale of the collaboration ruled out a quantitative evaluation, it would be decided whether the standard was suitable for roll-out to other Branch projects based on a qualitative measure: whether the standard received high ratings from Branch personnel as to usability and overall satisfaction.

The project used for piloting the Perspective-Based Inspection approach was a multi-mission framework designed for reuse. This was a good choice because key representatives from the three new missions would be involved in the inspections.

The perspective-based approach was applied to produce inspection procedures tailored for the specific quality needs of the branch. The technical information to do so was largely drawn through a series of interviews with Branch personnel. The framework team used the procedures to review requirements. The inspections were useful for indicating that a restructuring of the requirements document was needed, which led to changes in the development project plan.

The standard was sent out to other Branch personnel for review. Branch personnel were very positive. However, important changes were identified because the perspective of Attitude Control System (ACS) developers had not been adequately represented, a result of the specific personnel interviewed. Further iterations past the end of the collaboration resulted in draft Branch inspection standards for requirements and code which are on track to be baselined.

2.3 JPL: "Finding Defect Patterns in Reused Code" using Orthogonal Defect Classification

This effort used Orthogonal Defect Classification (ODC) to characterize defect reports for code that will be reused in mission-critical ground software. The application of ODC to NASA projects has been previously funded by SARP.

The goal was to identify patterns of defects prior to reuse of the code, and to successfully infuse ODC into a project. ODC, as adapted for NASA by the researchers, characterizes anomaly reports using four attributes: Activity, Trigger, Target, and Type.

There were several groups of players in this project: Software Quality Assurance (SQA), JPL's Software Quality Initiative (SQI), Dr. Robyn Lutz (JPL, Iowa State University), and of course the ground software project. Dr. Lutz worked with the project to customize the classification entries. The original idea was to have the project itself learn to do the classification and analysis of anomaly reports on the software. However, the funding for the collaboration was late, the project entered a busy period, and there was a JPL reorganization, so instead people in SQA and SQI were taught the technique, and, with help from the project, classified the anomalies. Dr. Lutz did the analysis and reported the findings to the project. Infusing ODC into the SQA and SQI organizations was an unexpected benefit of the collaboration.

The project was satisfied with the results of the ODC analysis. Though the ground software project was not continued, so the software was not reused, the software whose anomalies were analyzed was put into operation, and the analysis results were to be used to direct its maintenance. The same development team used on another project an ODC analysis which indicated process problems that the team had expected.

One factor that helps introduce the use of ODC is the use of a bug tracking database that is compatible with ODC classifications; for example, one that has pulldown menus so that the classification can be done easily when the anomaly is reported, rather than later when it is more difficult to decipher the anomaly report. Requirements for such a capability have been added to JPL's next generation problem reporting system.

2.4 JSC: "Can CodeSurfer Increase Inspection Efficiency?"

CodeSurfer is a commercial tool from Grammatech, Inc. for browsing C code. It provides lists of variables and constants used or set by functions, call graphs, pointer analysis, indications of dead code, etc. The objective of this project was for the Software Assurance organization to apply the tool during the inspection phase of an International Space Station (ISS) software component, to see if the tool made the inspections more time efficient and/or more productive; that is, more defects were found. Because the funding arrived late, and the acquisition took longer than anticipated, the window for the inspection phase of the module was missed. It was decided to apply CodeSurfer to the component anyway, as an experiment to compare with previous inspection results. Also, CodeSurfer was applied during inspection of another ISS component.

The results show that the time required for doing an inspection using CodeSurfer is reduced from that for a manual inspection, and the inspection is more productive. The collaboration's final report states that manual code inspection required 17 hours, and only 12.25 hours with CodeSurfer. Manual code inspection found 8 defects, whereas 18 (including 6 of the 8 found manually) were found using CodeSurfer. Though the defects were all classified as minor, these are clear benefits. However there were difficulties. There is a learning curve: the training helped, but the project suggested that the tool would be difficult to use if there was a long time between uses, so ideally, there should be people who use the tool more frequently. The tool required that the code compile with one of the compilers provided with the tool: this ran into problems because the code analyzed would only compile using a legacy compiler, so some adaptation was required. Also, Software Assurance did not always readily have all the required files. The vendor of CodeSurfer, GrammaTech, Inc., was responsive, but because of ITAR (i.e., export control) restrictions, the ISS code could not be sent to the vendor for their assistance. The net effect was that setup time swamped inspection time. Obviously, there is a learning curve, so setup time would be reduced in the future. The research infusion team sees these as generic problems to be dealt with for code analysis tools.

The summary impact is that the Software Assurance organization is evaluating continued use of CodeSurfer on C and C++ projects for reviews. They have demonstrated the tools to the engineers who developed the ISS components, and are interested in collaborating with other customers of Software Assurance in using the tool to troubleshoot software.

2.5 MSFC: "Static Analysis of Flight Software" using Coverity SWAT and C Global Surveyor

The objective of this effort was to apply two source code analysis tools to four flight software components, in order to find errors, and characterize the utility of the tools. The components varied in maturity from the coding and unit testing phase to the maintenance phase.

The two analysis tools were C Global surveyor (characterized above in Section 2.1) and Coverity, Inc.'s Software Analysis Toolset, SWAT (now called Prevent). The latter is a source code analysis tool for C programs that looks for certain types of errors, such as use of uninitialized variables, out-of-bounds indices (buffer overrun), dead code, and functions that should check their return value but don't. It does not claim complete coverage, in contrast to CGS, which does; that is, SWAT does not necessarily find all the errors of a particular type.

A team from Marshall Space Flight Center (MSFC) was trained at Ames Research Center (ARC) in the use of CGS. This resulted in a number of recommendations for the tool, similar to those found in the ARC collaboration (section 2.1). The tool produced about 300 warnings for a couple of the modules; about 20% were analyzed, and no errors were found. On the other hand, the technology developers reported that on one of the MSFC applications, CGS was 95% precise. An update to CGS that fixed some of the issues raised was delivered to MSFC, but it was not run again on their software.

The Coverity tool was applied to the components. It flagged a total of 74 errors in 14 minutes. Analysis of those errors by flight engineers resulted in no errors found in the most mature component, but 9 in the other components were considered errors that were registered to be fixed; four of these had escaped formal testing. A usability issue was brought to the attention of Coverity.

The project concluded that the Coverity SWAT tool thus had a low false alarm rate and fast execution times and was recommended for use in future projects' software development process if the associated licensing costs can be afforded.

2.6 USA: "USA Application of Perspective-Based Inspections"

The Perspective-Based Inspection approach was applied by Dr. Forrest Shull in an ISS software development project at United Space Alliance (USA). The goal was to increase the quality of the product, and increase inspection efficiency over previously used techniques.

Project personnel were interviewed to tailor the approach, and instruction was provided, with actual software inspected as part of the instruction. Defects were found during that inspection, which was surprising because that software was reused from a previous version and hence thought to be defect free. Following the course, Perspective-based inspections of code were carried out, finding a major defect which had escaped previous inspections. On a qualitative, subjective level, the response from the project team consisted of only positive comments.

The experience was that less time was required per inspector, who also had a more structured focus. It was noted that Perspective-Based Inspections required more inspectors than the project's usual practice. The approach was recommended as an optional practice at USA. A kit was created to easily help craft perspectives for smaller projects. The project recommended the approach for larger projects.

2.7 ARC: "Application of Software Cost Reduction (SCR) Tools and Methods to On-Orbit Crew Displays"

The SCR technology, originated by David Lorge Parnas, and further developed at the Naval Research Laboratory (NRL), provides tools and a method for developing, simulating, and analyzing formal requirements specifications. An SCR specification is represented in a tabular format (Parnas tables) and is based on a state-machine model. In addition to tools for consistency checking to detect syntax and type errors, missing cases, unwanted non-determinism, and tools to check application properties, such as safety properties, SCR also supports rapid construction of graphical user interfaces (GUIs) that simulate the target system's interface, allowing for simulation of the required system behavior based on the underlying SCR specification.

The goal of this project was to apply SCR tools and methods to develop and validate a requirements specification of the display interface to an incubator. The incubator was to be a Space Station Biological Research Project (SSBRP) science payload.

The SCR technology providers gave a three-day training course on the SCR tools and method at the NASA Ames Research Center (ARC) to the project members. Lack of availability of the tool on the preferred ARC platform at the time of the training meant that limited hands-on training occurred during that visit, though the tool was delivered shortly thereafter. Natural language incubator display requirements and use cases for its Flight mode were used as the basis for collaboration between the project members with the SCR technology providers. The technology providers encoded some of the requirements as a formal SCR requirements specification; this took about two person-weeks. The specification described behavior for setting the chamber fan speed and a goal temperature based on user inputs. The SCR technology providers had planned to give hands-on training on the GUI builder to the project members, but schedule conflicts prevented this. The SCR technology providers provided tutorial materials and remote assistance resulting in the construction of a customized GUI for the incubator display. The project members tested the constructed display GUI against the requirements and found its behavior consistent with the requirements.

The project members reported that no errors in the original natural language requirements and use cases were uncovered in this process, though the SCR technology providers noted there was a lack of completeness and existence of ambiguity in the original natural language requirements that was reflected in the SCR specification. An example of this is that it was not specified how the functions interacted or conflicted; e.g., what the required behavior should be when a new command is given before the previous command completes.

The project members considered the use of the SCR technology successful in that the SCR requirements specification correctly captured some requirements of the Incubator Display. It does not appear that the project members can develop SCR specifications unaided. The project members recommended, and the technology developers agreed, that the SCR methods and tools should be used when the understanding of the software requirements is mature. The project members concluded that the toolset is valuable for validating requirements prior to design, and made other recommendations regarding extensions to the SCR methods and tools which the technology providers said have been or could easily be implemented.

2.8 IVVF: "Infuse CodeSurfer into NASA IV&V Process"

As described above in Section 2.4, CodeSurfer is a commercial tool for browsing C and C++ code. It allows for visualization of data and control flow via, for example, call graphs, and forward and backward slicing. It was previously employed at JSC where it was used during code inspections (see Section 2.4).

The goal of the collaboration at NASA's Independent Verification and Validation Facility (IVVF) was to apply the tool to analyze flight code for IV&V. The original target software was not available in time, so software for a solar observatory was substituted. The observatory software was about 1.5 MB of C/C++ for C&DH, ACS, and instrument code. A delay was encountered by the tool not being able to ingest this software; this was eventually repaired in a new release of CodeSurfer. The observatory software analysis task was transitioned to another contractor which ended the collaboration. To add value to the collaboration, GrammaTech provided the infusion effort with the results of running its CodeSonar defect detection tool on the observatory software. Because of the various changes in the collaboration, rigorous time and effectiveness comparisons with other tools and previous experience could not be obtained.

CodeSonar identified several defects not identified by other tools or manual analysis. It reported about 60 defects and had a false alarm rate of about 50% which was in line with expectations. The reported defects were analyzed using another tool, Understand for C++, to determine whether they were true defects or false alarms; this took about half an hour per defect, which would have been less if the integrated CodeSurfer/CodeSonar interface (which exists but was not provided) had been used.

The project suggested that the people who will set up CodeSurfer to ingest the target software need to be familiar with compiler technology, and receive separate training, but users unfamiliar with compiler technology can readily become proficient in using CodeSurfer once it is set up on the target software and they are familiar with the platform. The project said ease of adoption was enhanced by using the Unix version of CodeSurfer.

The project recommended the continued use of CodeSonar, especially with the integrated interface with CodeSurfer. It also recommended CodeSurfer when the control and data flows are sufficiently complex, and the incurred setup time doesn't swamp the analysis time.

Despite changes in the prime IV&V contractor, CodeSurfer resides in the IVVF tools lab, and it is being used on another project.

2.9 JPL: Application of SpecTRM at JPL's Advanced Project Design Team (TeamX)

SpecTRM is a tool, from Safeware Engineering Corporation, that provides for capture of requirements, assumptions, design, design principles, design rationale, hazards, risks and their linkages.

The Jet Propulsion Laboratory (JPL) created the Advanced Projects Design Team (Team X) in April 1995. This team produces conceptual designs of space missions for the purpose of analyzing the feasibility of mission ideas proposed by its customers. The customers often consist of principal investigators of design teams who aim to plan new mission proposals. The study takes one to two weeks (usually involving 3 3-hour collaborative sessions) and the design is then documented in a 30 to 80-page report that includes equipment lists, mass and power budgets, system and subsystem descriptions, and a projected mission cost estimate. The study is then reviewed and summarized and an abbreviated report is also produced. There have been over 100 to date.

Historically, rationale for design options and their risks have not been retained during the fast-paced Team X design sessions so it is not possible (for example) to subsequently investigate the sensitivity of the design to changes in the design parameters. The goal of this infusion was to investigate the feasibility and benefit of using SpecTRM during Team X session to capture design rationale (options considered, the basis for making design decisions, and the hazards and risks associated with the decisions), to estimate the benefits of doing so, and to determine the changes needed to accommodate SpecTRM's use if it were decided to be beneficial.

This infusion used an aerobot mission to Titan as its TeamX test case. The process carried out was for members of TeamX to provide the technology provider with information about their work during the design session, so that the provider could enter the data into SpecTRM. (This process was followed since purchase and training in the use of SpecTRM was not included in the proposal.) The provider organized the information in SpecTRM as system-level goals, requirements, assumptions, constraints, design principles, action items, hazards, and then linkages among them. The data captured was a subset of the information captured in the TeamX directory. The project claimed that the biggest benefit was that these attributes were systematically described and traceable.

The project described conditions and suggestions for the integration of SpecTRM into the TeamX process. For example, it would help to have a knowledge base of previous designs and their rationales. Also, the systems engineer on TeamX should be trained in the tool. Another suggestion is to build an interface to SpecTRM that provides TeamX members the same format for entering information as they use now; also suggested was a concurrent, multi-user SpecTRM.

A journal paper describing the SpecTRM/TeamX experience is in progress. The technology assistant was hired at JPL, so expertise in SpecTRM will be readily available at JPL.

As far as adopting SpecTRM, the TeamX management will decide what its priorities are and how much funding to allocate to each priority. If it turns out that design rationale capture is a priority and funding is allocated to it, SpecTRM is one of the options TeamX will consider.

3. Success Criteria

From the outset, it was our desire that the long-term success criteria would be that the research products used in the collaborations become adopted for future software development by the proposing teams and/or their organizations. The need for patience ("long term success") stems from the fact that we are often dealing with mid TRL-level (Technology Readiness Level) research products that may lack productization, and hence will require further development before being ready for mainstream adoption by flight projects. Even for high TRL products there are factors that constrain immediate adoption. For example, a high TRL commercial product may have a high license fee, accommodation of which requires advance budgetary planning. The timescale of our efforts motivated us to seek several short-term success criteria that would be indicative of progress towards our long-term one, as follows:

1. The success criteria of the collaboration projects funded are met. This includes a positive rating for each product on the evaluation criteria metric.
2. The research product is adopted by the collaborating software development team for current use.
3. The research product is adopted by the collaborating software development team for current use
4. The software development team using the product provides feedback, including performance data, to the research team to guide future development of the product.
5. Six months after the funded collaboration period the research product is still being used by the development project or by a successor development project.
6. Independent of the success of the collaborations, "lessons learned" regarding the challenges and success factors for software development technology infusion within NASA.

Also relevant to judging the impact of the collaborations is the penetration factor (PF) used for Software Assurance Research Program quarterly reviews. Only the two highest PFs are of interest to us in the research infusion initiative:

PF 8: Data passed back to project;

PF 9: Results actually used by the project.

Table 1 summarizes each of the nine infusions that have completed as of the time of writing. It shows the penetration factor of each project, a tick (✓) in the relevant column indicates that corresponding criterion (1 to 6) above is satisfied. A clock symbol (🕐) indicates that it is anticipated that this criterion will be satisfied shortly (within the 2006-2007 timeframe). A star (★) indicates that the criterion will be achieved only conditionally on the cost of the tool.

Project	PF	1	2	3	4	5	6	Impacts
ARC - **CGS on ISS payload software**	**9**				✓		✓	**Found 2 errors to be fixed. Useful feedback to the CGS developers.**
GSFC - **PBI in Flight Software Branch**	**9**	✓	✓	✓	✓	✓	✓	**PBI led to changes in the projects' development plan. Expected rollout of PBI in FSB standards.**
JPL - **ODC on ground software**	**9**	✓		✓	✓		✓	**Training occurred in several JPL organizations. ODC led to several recommendations that will be used in project maintenance phase.**
JSC - **CodeSurfer for Inspections of ISS software**	**9**	✓	✓		✓		✓	**Found 12 additional (minor) defects. Tool is continuing to be evaluated.**
MSFC - **SWAT & CGS on Flight Software**	**9**	**S**	★	★	✓	🕐	✓	**Useful feedback to the CGS developers. SWAT found 9 defects worth fixing in the software, some of which had escaped formal testing.**
USA - **PBI on ISS Software**	**9**	✓	✓	✓	✓	✓	✓	**Found 6 "major" defects, several of which had escaped previous inspections and/or occurred in reused code. Will continue to be used and was recommended as an optional process.**
ARC - **SCR on ISS payload software**	**8**	✓			✓		✓	**Good exposure to the technology.**
IVVF – **CodeSurfer/Sonar on Flight Software**	**9**	✓	✓	✓	✓	🕐	✓	**CodeSonar found non-trivial defects, and its use is recommended.**
JPL - **SpecTRM to capture mission design rationale**	**8**	✓			✓		✓	**Adoption considerations were explored. Journal article in progress.**

4. Some Lessons Learned

The completed research infusion projects that have completed have raised a number of issues, confirmed some expectations, and debunked others:

- Some developers are not proficient at research-oriented activities and need guidance and oversight. These teams are likely to benefit from more detailed *pro forma* documentation or templates (kick-off meeting agenda, project plan, reports). For specific categories of tools (such as source code analysis tools) we can provide very detailed templates. They also require frequent oversight (a) to ensure that communication is occurring between developers and technology vendors and (b) to ensure that the schedule is being followed. Not all the projects require this level of support but it is likely to benefit Research Infusion by promoting uniform, higher-quality collaboration practice.
- There are various answers to the question "What is the next step"—from research infusion to technology transfer. A general solution is unlikely. Some technologies are readily integrated and generalized into a parent organization's existing processes (for example, Perspective-based Inspections at GSFC)—they are modifications to existing processes. Various other technology-specific approaches may be appropriate; e.g., PBI may be supported by the Software Engineering Initiative's Training strategy.
- Tighter qualification of technology/project combination may be needed. One of the source code analysis tools used at ARC and MSFC had previously been successfully applied to NASA software. However, the software that was the subject of the infusion study had different technical features to the previously successful software applications, and it turned out that the analysis tool did not transition well to the software with different features. Also, the appropriate lifecycle context and purpose for the tool (in this case) may not have been clear to the development teams.
- Sometimes project personnel already have in mind technologies they are interested in and the research infusion effort serves predominantly to provide seed money so that the desired collaboration can take place, and track its progress once initiated.. This was the case with the JPL/SpecTRM collaboration. In the case of an ongoing collaboration, the research infusion team brokered the collaboration between the developers and technology providers.
- Collaborations' project plans should explicitly include an iterative approach to technology application, scaling up with each iteration.
- Leading-edge tools sometimes have problems, e.g., needing specialized skills for set-up. Technology providers have made efforts to compensate.
- To succeed, training and continued support are needed. For example, USA received onsite training on applying PBI technology to its own application. This reduced risk and cost as well, since part of the target application was used in the training session. "The most successful way to do tech transfer is to put a member of the [technology vendor team] on the development team"[1]
- The profile of effort required to learn new technologies varies with the technology. For example, a few days may be enough to learn a software browsing tool such as CodeSurfer, or to apply SCR tools to an existing SCR specification. But committed blocks of time and more resources and suitable background may be required to become facile with aspects of the technologies, such as SCR specification development, with the expectation that the payoff (such as being able to take advantage of applying the SCR tools) would be worth the effort.
- Busy researchers and project members may have scheduling pressures that take precedence over infusion studies, which may lead to significant delays in the infusion projects.
- NASA is a dynamic environment. It is important to consider the loss of organizational memory as a risk up front and plan for its mitigation. The application that SCR was applied to was stopped, and its employees dispersed, so expertise in SCR was dispersed as well. The contractor PI using CodeSurfer lost its prime status so work on analyzing the solar observatory was transferred to a new contractor not necessarily using CodeSurfer; however in this case, the tool remains in the IVVF tools lab and is still in use at the original PI contractor on other projects.
- If the lead-time between technology proposal and beginning the project is too great, the necessary personnel may be lost, or it may be impractical, or unbeneficial, to use the technology at this stage of the development. We have had two projects which needed to change and use alternative software, and another project which it was no longer feasible to run as a result of this delay. In an attempt to counteract this, the 2007 process for soliciting proposals and

[1] Matt Barry, JPL, (paraphrased) communication to the authors.

choosing among them will begin later in the year, nearer to the time that funding will be available.

5. Conclusions

The overall impact and benefits of research infusion to space systems are several: previously-inaccessible software assurance technologies have been successfully infused; some have been adopted for inclusion in organizations' development practice; several have continued to be used for some time following the end of the collaboration; the software development team has provided feedback to the technology developers; and, lessons learned have been identified regarding the challenges and success factors for software development within NASA.

Overall, Research Infusion's set of completed collaborations supports the hypothesis that with selection of appropriate technologies, matching of technologies with software development teams, and guidance and oversight, infusion of new software engineering technologies can be performed successfully on a minimal budget. Note however that the technologies considered in these efforts have been constrained to those that can be introduced within the context of existing software development practices. For technologies whose infusion would be more revolutionary, requiring a radical shift in existing practices (e.g., an approach that requires formal specification of the entire software system, or a new programming language that is incompatible with existing platforms and personnel skills), significant additional factors that we have not had to address will likely be involved.

6. Acknowledgements

The research described in this paper was carried out at NASA Ames Research Center, Langley Research Center, Marshall Space Flight Center, and Goddard Space Flight Center, and at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

The research infusion team was led by Tom Pressburger from 2003 to 2005. Mike Hinchey now leads the team. When this work was undertaken, team members included the authors, along with Ben Di Vito (Langley), Luis Trevino (Marshall), and Tim Menzies (West Virginia University); Wes Deadrick (NASA IV&V Facility) has been an advisor and reviewer of collaborations. Current team members are: Mike Hinchey (Goddard), Tom Pressburger (Ames), Ken Chen (Johnson), Caroline Wang (Marshall), Michael Holloway (Langley) and Allen Nikora (JPL).

We are indebted to John Kelly, NASA Office of the Chief Engineer, who leads the NASA Software Working Group and provides support for the initiative. Martha Wetherholt in the NASA Office of Safety and Mission Assurance also provides financial and other support for the initiative, administered in part by the NASA IV&V Facility, where the Software Assurance Research program operates under the direction of Marcus Fisher (previously under Ken McGill).

We would like to acknowledge the many researchers who have lent their support and the many software developers who have submitted collaboration proposals, and worked with us to make this initiative such a success.

References

[1] T. Pressburger, B. Di Vito, M.S. Feather, M.G. Hinchey, L. Markosian and L. Trevino. Infusing Software Assurance Research Techniques into Use. *Proc. 2006 IEEE Aerospace Conference,* Big Sky, MT, 10-14 March 2006, IEEE Computer Society Press.

[2] M.G. Hinchey, T. Pressburger, L. Markosian and M.S. Feather. The NASA Software Research Infusion Initiative: Successful Technology Transfer for Software Assurance. *Proc. TT'06, Workshop on Technology Transfer for Software Engineering, International Conference on Software Engineering,* Shanghai, China, 20-28 May 2006, ACM Press.

For more information on each of the completed infusion collaborations discussed herein, and the additional ones ongoing at the time this paper was written, see the website
http://ti.arc.nasa.gov/researchinfusion/